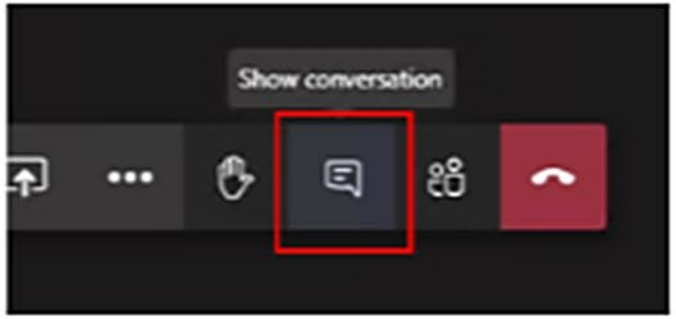
Welcome to the R/RStudio Workshop

November 9th, 2022

Teams Best Practices



- *Mute yourself and use chat, click the Teams Show conversation button*
- ***When using Teams audio use the Mute button** to mute and unmute yourself*

  The mute button looks like this when muted

  The mute button looks like this when unmuted

- ***If you are using a phone**, use *6 to mute and unmute yourself*
- *Please use the chat feature to ask questions, the speaker or a moderator may ask you to unmute yourself to elaborate or discuss your question*

United States® Census Bureau

Participation Reactions:

Workshop Team

Census Team:

- Jessica Klein, ERD
- Cecile Murray, ERD
- John Lombardi, EWD
- Keith Savage, EID

Center for Applied Technology Team:

- Christopher Jackson, CAT Program Manager
- Kevin Schweickhardt, Operations Manager
- Mohammed Chizari, Data Scientist
- Saleem Shaik, Data Scientist
- Jimmy Pazouki, Systems Administrator
- Llewellyn Forbes, Operations Manager

Department of Commerce
U.S. Census Bureau
Room 1J250
4600 Silver Hill Road
Suitland, MD 20746
(301) 763 - 4300
cat@census.gov

Workshop Goals and Expectations

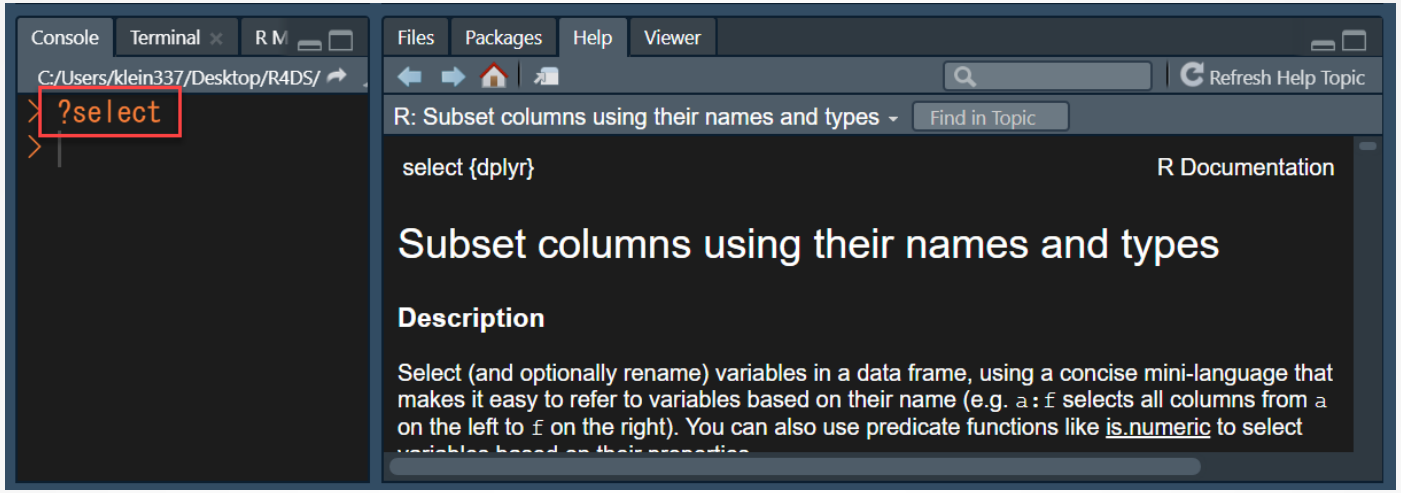
- This workshop is designed to give attendees exposure to a valuable tool that can be used for data science work.
- This workshop will not train you to be a data scientist but will introduce you to data science tools to help innovate your data work. With some time, patience and a lot of hands-on practice, you too can learn how to effectively use these tools to explore your data.
- Please support each other by being patient and encouraging. There is no room for negativity in this workshop.
- Use the Teams help page for troubleshooting and assistance; time is short so the more we can focus on the workshop content and exercises, the better.
- If you leave this workshop wanting to know more, you are encouraged to seek out training opportunities in your division. There is a government wide IT modernization effort in effect to upskill staff, which includes funding, and the agency wants you to have these valuable skills.

United States® Census Bureau

R

Troubleshooting and getting help

- The fastest way to get assistance on a function is to look it up from your console using ?function



- Use google search and stack overflow for assistance. Chances are someone else has encountered your problem and has a solution. Copy the exact problem or error into google and see what resources are returned.

- If you need technical assistance during the workshop, raise your hand and ask for help via breakout room. Kevin S will put you in a breakout room with a member of the R tech team. Please include the exact error or reproducible code example for easy of troubleshooting. Please also contact the cat@census.gov for software and installation assistance.

  - R: Getting Help with R (r-project.org)

More Resources for

- RStudio Cheat sheets – RStudio: Cheat sheets for popular packages
- RStudio Education: Trainings curated by RStudio employees
- Percipio: Our CLC data science learning platform
- Learn R, Python & Data Science Online | DataCamp: Free Intro to R and Python Classes, cheap for new members to get a year pass- good use of training funds!
- Welcome | R for Data Science (had.co.nz): All content from this workshop came from this book. It is a fantastic resource. Please check it out.
- Pluralsight | The tech workforce development company: Video tutorials on R, Python and more!
- swirl: Learn R, in R. (swirlstats.com): Interactive hands-on lessons within RStudio

United States® Census Bureau

- Schedule some time with the CAT for one-on-one instruction: cat@census.gov
- Join the CBRUG group; our Teams site holds office hours, maintains resources and has many friendly users to collaborate with

What are R and RStudio?

- R is a language for statistical computing and graphics. The R environment is an integrated suite of software packages used for data manipulation, calculation and visualization. It includes
  - an effective data handling and storage facility, where data does not leave the hard drive for computation or storage
  - a suite of operators for calculations on arrays, in particular matrices,
  - a large, comprehensible, integrated collection of tools for data analysis,
  - graphical facilities for data analysis and display either on-screen or on hardcopy, and
  - a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.
- RStudio is a free and open-source integrated development environment (IDE) for R, a programming language for statistical computing and graphics
  - Using R and RStudio | Hands-On Programming with R (rstudio-education.github.io)

United States® Census Bureau

R

RStudio User Interface:

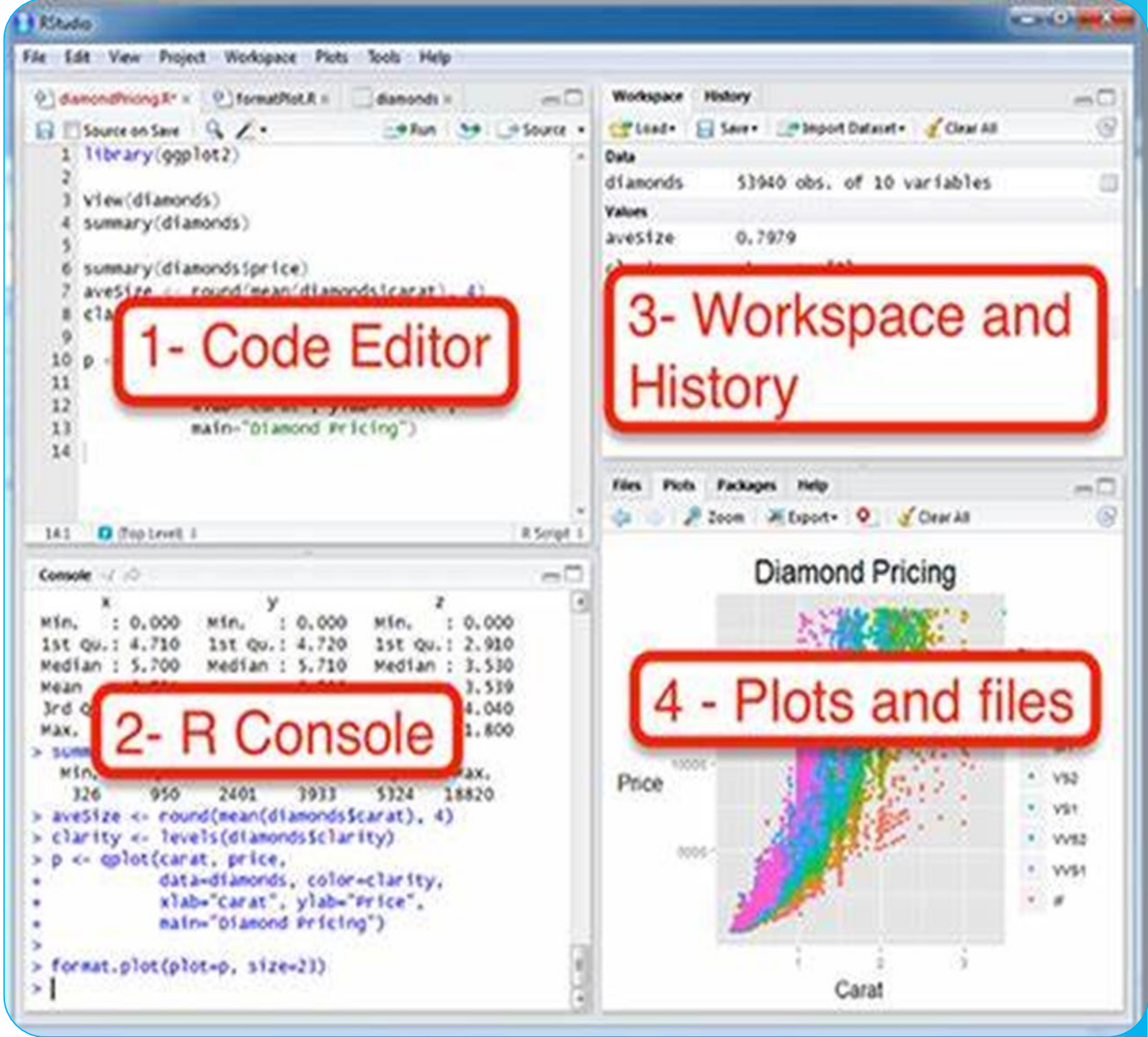
1) Code editor (notepad)
2) R Console (output)
3) Workspace and history
4) Plots and files (visualization)

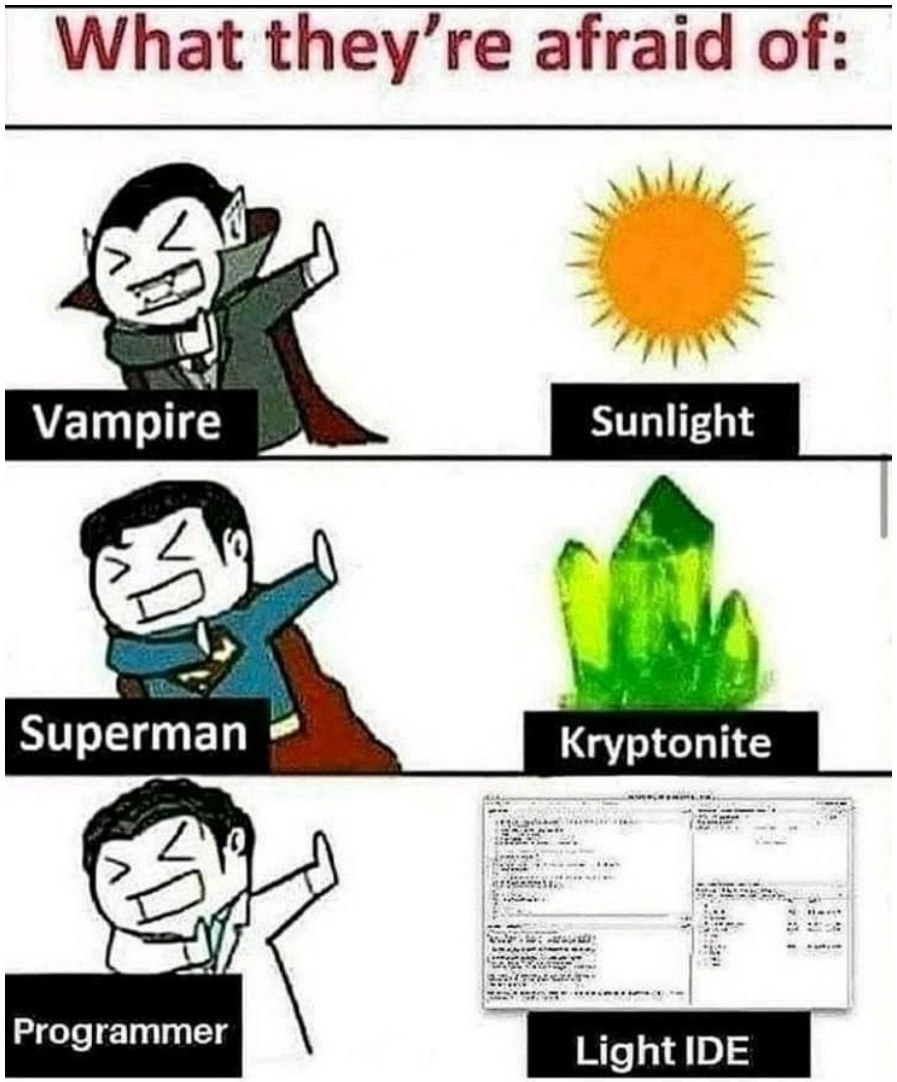
RStudio Video Tour:
RStudio 1.4 - A quick tour – RStudio

RStudio Community:
(6) RStudio Community

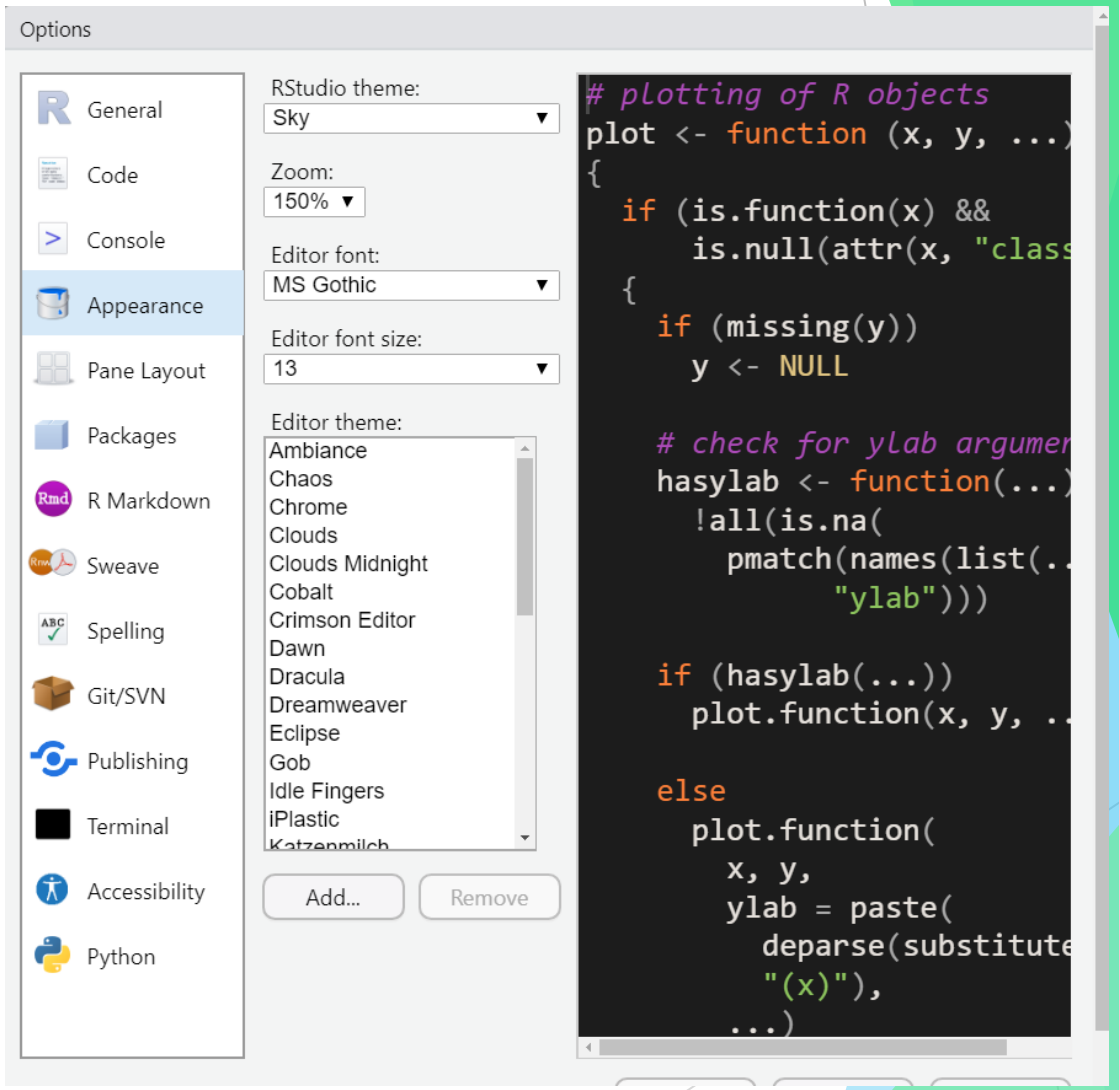
RStudio Cheat Sheets:
Cheatsheets – Posit

Keyboard Shortcuts:
alt + shift + k



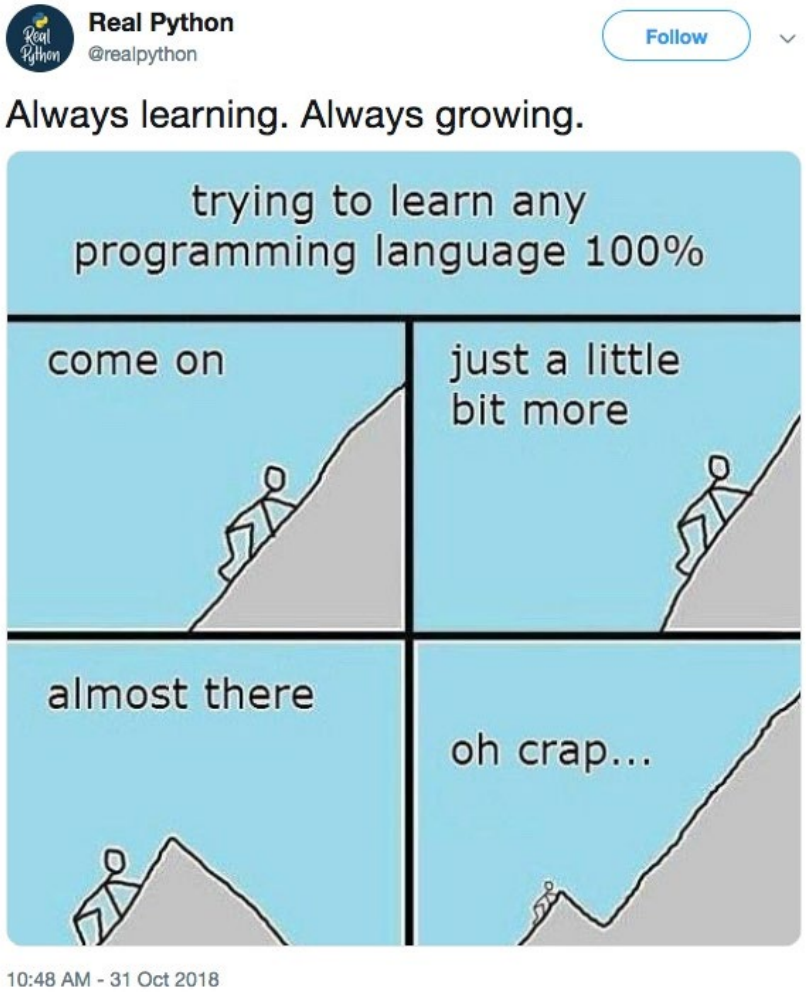


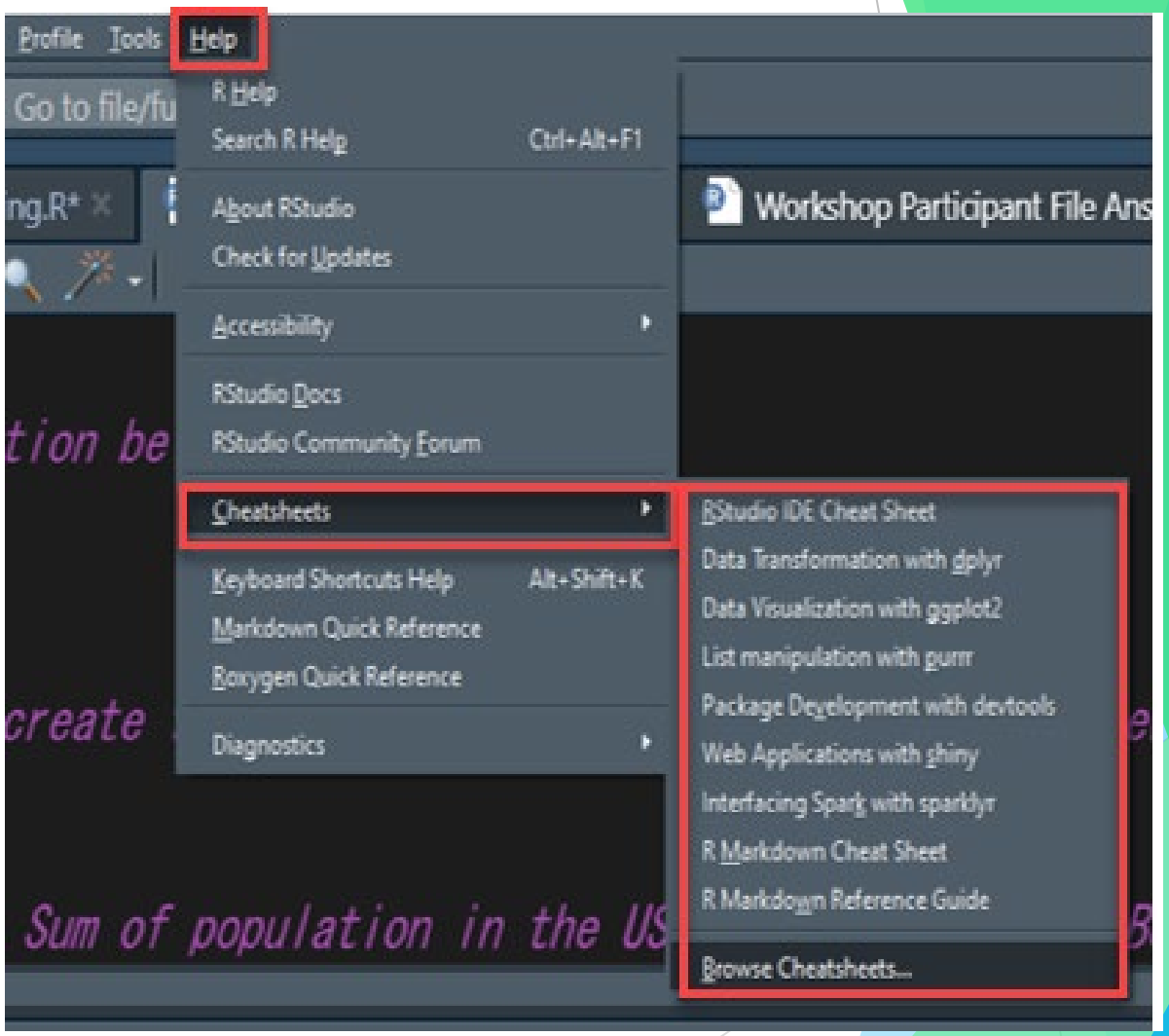
Change appearance to dark



United States® Census Bureau

Cheatsheets!





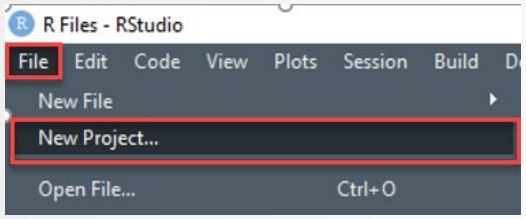
Print these out as helpful resources

United States® Census Bureau

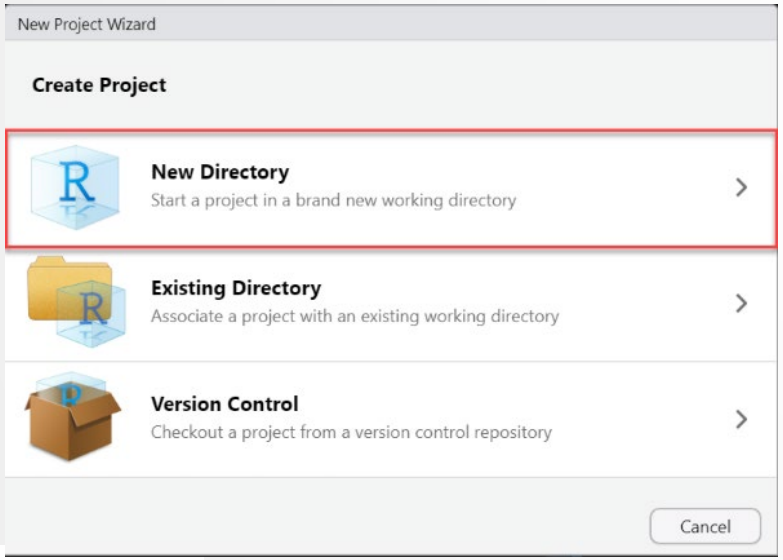
Get your data: start a new R project

- Open a new project and create a new directory

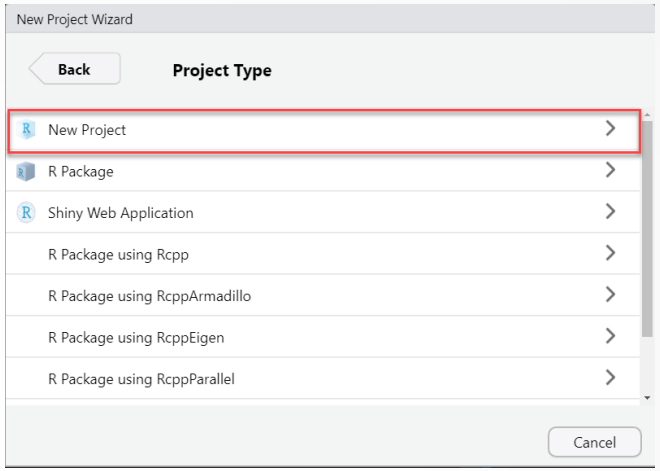
1



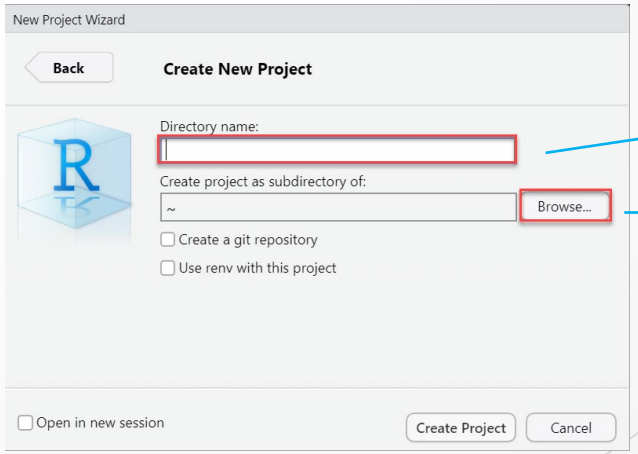
2



3



4



5

Save to new directory:

- Four .R files
- Three data files
  - 2 .csv
  - 1 .xlsx

Directory name:
R4DS

Browse to:
C:/Users/jbidXXX/Desktop

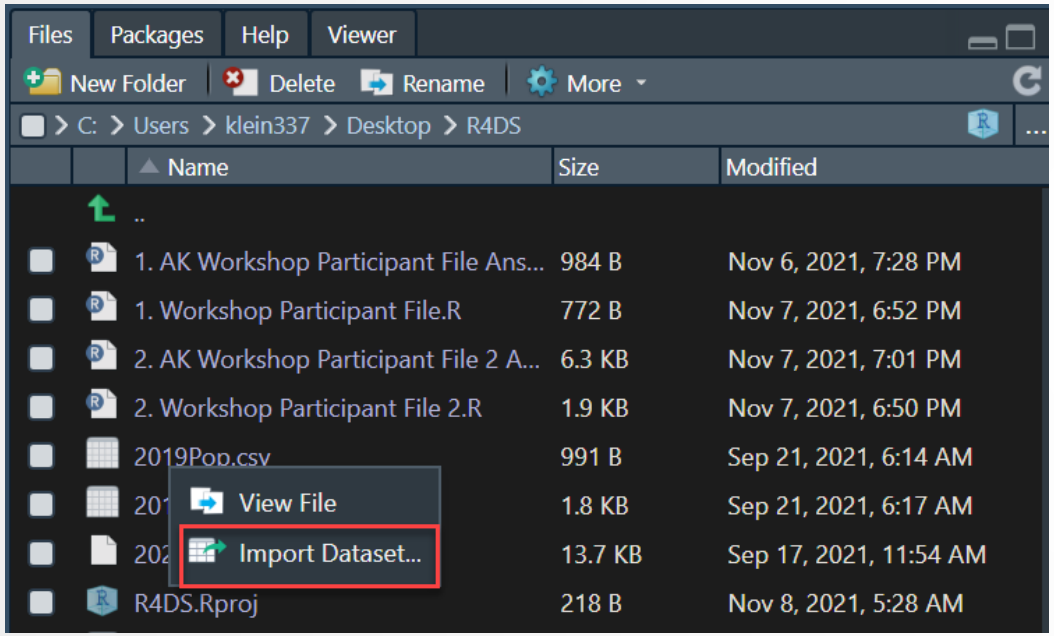
Get your data: change your working directory

- Change your working directory
  - setwd("C:/Users/JBIDXXX/Downloads")
  - list.files()
- Read in data
  - FileName <- read_excel(“filename.xlsx”)
    - Census2020 <- read_excel("2020 Census File.xlsx“)
    - Must have loaded the readxl package
  - FileName <- read.csv(“filename.csv”)
    - Poverty2019 <- read_csv("2019Poverty.csv")

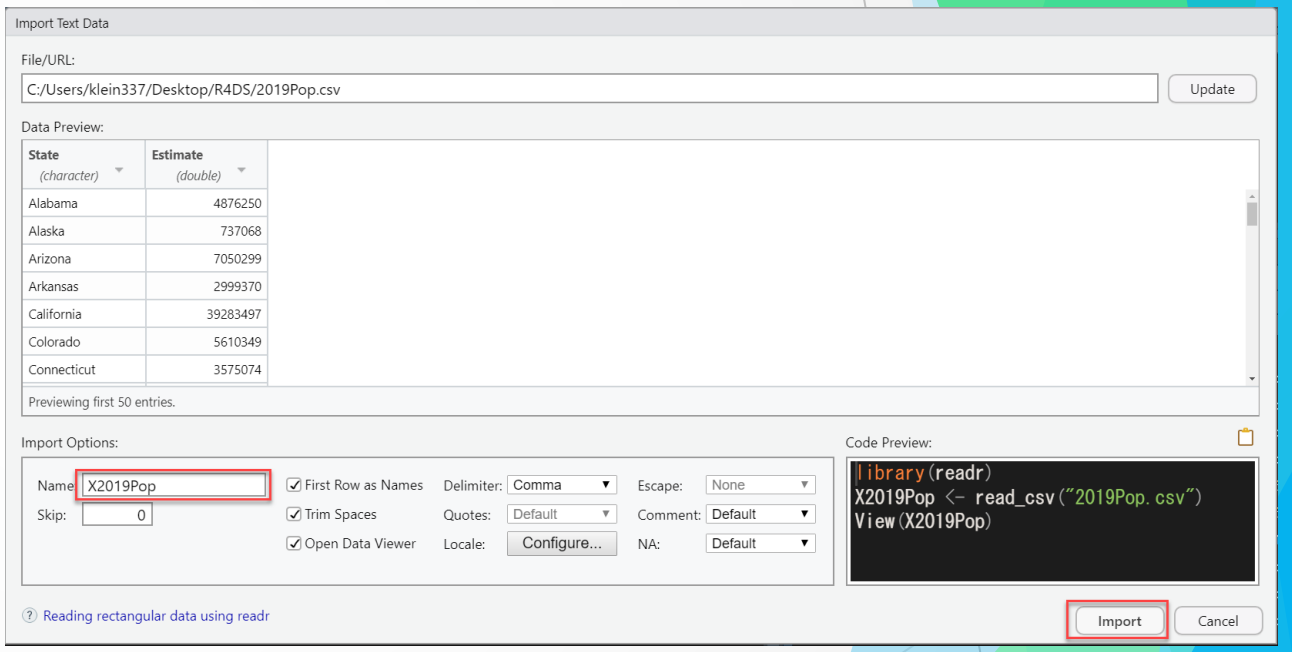
United States® Census Bureau

Import data from Files pane

1 Click on data name, select import dataset



2 Change dataset name, and select import



Change dataset names from-to:

- X2019Pop to Census2019
- X2019Poverty to Poverty2019
- X2020 Census File to Census2020

Helpful Commands to Inspect our Data

- str(): Structure of dataframe, including head, dimensions and columns
- glimpse(): See the columns of the dataframe and display a portion, similar to str()
- dim(): Dimensions of dataframe
- head() and tail(): Look at the top and bottom 6 rows of the dataset, including column names
- colnames() or names(): Displays the column names of the dataframe, each in a different format
- rownames(): Displays the rownames of the dataframe
- min() and max(): Will return the smallest and largest values in the column
- summary(): Displays summary statistics of each column in a dataframe, best for numerical data
- View(): Opens complete dataset in new window

- To identify a column within the dataset: df$colname
- To identify an exact position within the dataset: df[row,col]
- Use the assignment operator to assign values/store data to an object: <-

United States® Census Bureau

R

Importance of Data Manipulation

- To start your data science journey, you need to be comfortable with data manipulation. Data manipulation is the task of reformatting and transforming your data into a structured and readable dataset. Data manipulation is the most fundamental data science skill, and to move forward with endeavors such as machine learning and advanced visualization, you must be comfortable with performing these tasks to make your data more organized and readable for analysis. The cleaner your data going in, the more useful it will be for analysis and later visualization.

Source: 5 Data transformation | R for Data Science (had.co.nz)

5.1.3 dplyr basics

In this chapter you are going to learn the five key dplyr functions that allow you to solve the vast majority of your data manipulation challenges:

- Pick observations by their values (`filter()`).
- Reorder the rows (`arrange()`).
- Pick variables by their names (`select()`).
- Create new variables with functions of existing variables (`mutate()`).
- Collapse many values down to a single summary (`summarise()`).

These can all be used in conjunction with `group_by()` which changes the scope of each function from operating on the entire dataset to operating on it group-by-group. These six functions provide the verbs for a language of data manipulation.

All verbs work similarly:

1. The first argument is a data frame.
2. The subsequent arguments describe what to do with the data frame, using the variable names (without quotes).
3. The result is a new data frame.

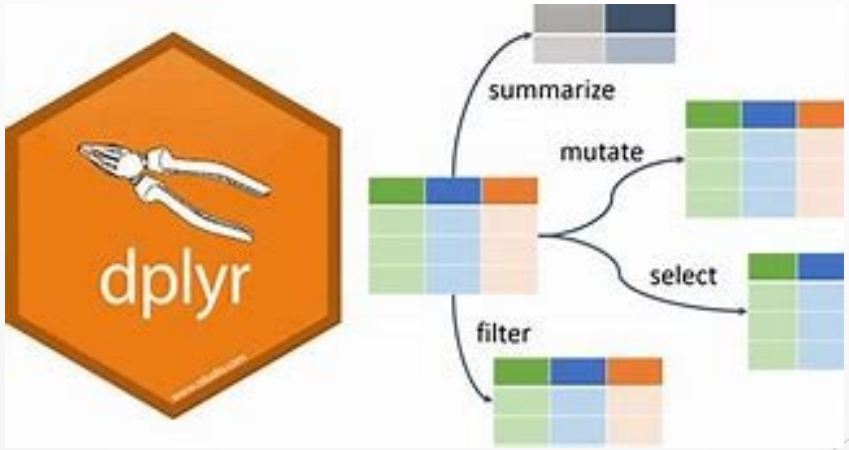
Together these properties make it easy to chain together multiple simple steps to achieve a complex result. Let's dive in and see how these verbs work.

United States® Census Bureau

R

dplyr overview and loading packages

- Function reference • dplyr (tidyverse.org)
- dplyr is a core package within the tidyverse ecosystem. dplyr has many functions available for data manipulation that are structured in an intuitive, user-friendly manner. The primary application of this package is to transform existing datasets into a format better suited for analysis and visualization.
- There are two ways to pass in arguments:
  - Pass in the dataframe as well as the information: newdf <- select(df, col_1, col_2)
  - Chaining is used %>% to put multiple functions together: newdf <- df %>% select(col_1, col_2)
    - We will demonstrate chaining in this workshop



United States® Census Bureau

R

select(): keeps/drops variables based on col names

- Subset columns using their names and types — select • dplyr (tidyverse.org)
- Often we will have more columns than needed, and that makes working with the dataset cumbersome. The solution is to keep or drop column names using select()
  - Newdf <- df %>% select(colname1, colname2…) This will keep columns
  - Newdf <- df %>% select(-c(colname1, colname2…)) This will drop columns
- Exercise goal: Remove the columns from the df: Census2020 that are not relevant to 2020. Keep colnames: “Area”, “Numeric Change”, “2020 Census Resident Population”, “Percent Change”, “State Rank based on 2020 Census Resident Population”
- See here for select() helpers: Select helpers — select_helpers • tidyselect (r-lib.org)

select(): keeps/drops variables based on col names

- Code, both will leave the same result:

```
Census2020Sub <- Census2020 %>%
  select(`Area`,
         `2020 Census Resident Population`,
         `Numeric Change`,
         `Percent Change`,
         `State Rank Based on 2020 Census Resident Population`)

Census2020SubOpt2 <- Census2020 %>%
  select(-c(`2010 Census Resident Population`,
            `State Rank Based on 2010 Census Resident Population`:
              `State Rank Based on Percent Change`))
```

- Check solution:

```
colnames(Census2020Sub)
## [1] "Area"
## [2] "2020 Census Resident Population"
## [3] "Numeric Change"
## [4] "Percent Change"
## [5] "State Rank Based on 2020 Census Resident Population"

colnames(Census2020SubOpt2)
## [1] "Area"
## [2] "2020 Census Resident Population"
## [3] "Numeric Change"
## [4] "Percent Change"
## [5] "State Rank Based on 2020 Census Resident Population"
```

rename(): rename existing columns

- Rename columns — rename • dplyr (tidyverse.org)
- rename allows us to rename the existing columns to a meaningful name. Having simpler column names makes data manipulation easier, and rename will accomplish this
  - Newdf <- df %>% rename(newname = old_name, newname2 = old_name2,…)
- Exercise goal: Rename columns from df: Census2020Sub to simpler names
  - Rename:
    - State = Area
    - Pop2020 = `2020 Census Resident Population`
    - NumChange2020 = `Numeric Change`
    - PercentChange2020 = `Percent Change`
    - StateRank = `State Rank Based on 2020 Census Resident Population`

United States® Census Bureau

R

rename(): rename existing columns

- Code:

```
Census2020Sub <- Census2020Sub %>%
  rename(State = Area,
         Pop2020 = `2020 Census Resident Population`,
         NumChange2020 = `Numeric Change`,
         PercentChange2020 = `Percent Change`,
         StateRank = `State Rank Based on 2020 Census Resident Population`)
```

- Check Solution:

```
str(Census2020Sub)
## tibble [51 x 5] (S3: tbl_df/tbl/data.frame)
##  $ State             : chr [1:51] "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ Pop2020           : num [1:51] 5024279 733391 7151502 3011524 39538223 ...
##  $ NumChange2020     : num [1:51] 244543 23160 759485 95606 2284267 ...
##  $ PercentChange2020: num [1:51] 5.1 3.3 11.9 3.3 6.1 14.8 0.9 10.2 14.6 14.6 ...
##  $ StateRank         : chr [1:51] "24" "48" "14" "33" ...
```

United States® Census Bureau

R

bonus1: combine select() and rename()

- Challenge: In one command, from the original dataset, select the Area, 2020 Population and 2010 Population variables. Rename them to easier names State, 2020Pop and 2010Pop.

- Code:

```
Census2020Bonus1 <- Census2020 %>%
  select(`Area`,
         `2020 Census Resident Population`,
         `2010 Census Resident Population`) %>%
  rename(State = Area,
         Pop2020 = `2020 Census Resident Population`,
         Pop2010 = `2010 Census Resident Population`)
```

- Check Solution:

```
glimpse(Census2020Bonus1)
## Rows: 51
## Columns: 3
## $ State   <chr> "Alabama", "Alaska", "Arizona", "Arkansas", "California", "Col~
## $ Pop2020 <dbl> 5024279, 733391, 7151502, 3011524, 39538223, 5773714, 3605944,~
## $ Pop2010 <dbl> 4779736, 710231, 6392017, 2915918, 37253956, 5029196, 3574097,~
```

Take a Break, 10 minutes

United States® Census Bureau

filter(): subset rows using column values

- Subset rows using column values — filter • dplyr (tidyverse.org)
- filter allows us to subset our dataset to only contain records, or rows, that are relevant to our data science project
- There are several functions and operators used to construct the filter expression, the most common for our purposes will be

- `==` (Equal to)
- `!=` (Not equal to)
- `<` (Less than)
- `<=` (Less than or equal to)
- `>` (Greater than)
- `>=` (Greater than or equal to)

  - Newdf <- df %>% filter(colname == “criteria”)

- Exercise goal: Create two new datasets from df: Census2020Sub; and 1) Filter df by population greater than 9,999,999 in 2020 2) Filter df by population less than or equal to 9,999,999

filter(): subset rows using column values

- Code:

```
PopAboveLimit <- Census2020Sub %>%
  filter(Pop2020 > 9999999)

PopBelowLimit <- Census2020Sub %>%
  filter(Pop2020 <= 9999999)
```

- Check Solution:

```
dim(PopAboveLimit)
## [1] 10  5
dim(PopBelowLimit)
## [1] 41  5
```

- Extra: to build “and” or “or” condition, use “&” or “|” respectively:

```
PopAboveLimitAND <- Census2020Sub %>%
  filter(Pop2020 > 9999999 & StateRank >= 9)

PopAboveLimitOR <- Census2020Sub %>%
  filter(Pop2020 > 9999999 | StateRank >= 50)
```

- Check Solution:

```
dim(PopAboveLimitAND)
## [1] 1 5
dim(PopAboveLimitOR)
## [1] 12  5
```

arrange(): change the ordering of the rows

- Arrange rows by column values — arrange • dplyr (tidyverse.org)
- arrange allows us to arrange our rows of data by the values of a selected column
  - Newdf <- df %>% arrange(colname)
  - Newdf <- df %>% arrange(desc(colname))
- Exercise goal: Arrange the two new df: PopAboveLimit and df: PopBelowLimit by both ascending and descending order on StateRank. This will result in four new datasets.

United States® Census Bureau

R

arrange(): changes the ordering of the rows, ascending

- Code:

```
TopPopAsce <- PopAboveLimit %>%
  arrange(StateRank)

LowPopAsce <- PopBelowLimit %>%
  arrange(StateRank)
```

- Check Solution:

```
head(TopPopAsce)
## # A tibble: 6 x 5
##   State         Pop2020 NumChange2020 PercentChange2020 StateRank
##   <chr>           <dbl>         <dbl>             <dbl> <chr>
## 1 California   39538223       2284267               6.1 1
## 2 Michigan     10077331        193691               2   10
## 3 Texas        29145505       3999944              15.9 2
## 4 Florida      21538187       2736877              14.6 3
## 5 New York     20201249        823147               4.2 4
## 6 Pennsylvania 13002700        300321               2.4 5

head(LowPopAsce)
## # A tibble: 6 x 5
##   State         Pop2020 NumChange2020 PercentChange2020 StateRank
##   <chr>           <dbl>         <dbl>             <dbl> <chr>
## 1 New Jersey    9288994        497100               5.7 11
## 2 Virginia      8631393        630369               7.9 12
## 3 Washington    7705281        980741              14.6 13
## 4 Arizona       7151502        759485              11.9 14
## 5 Massachusetts 7029917        482288               7.4 15
## 6 Tennessee     6910840        564735               8.9 16
```

arrange(): changes the ordering of the rows, descending

- Code:

```
TopPopDesc <- PopAboveLimit %>%
  arrange(desc(StateRank))

LowPopDesc <- PopBelowLimit %>%
  arrange(desc(StateRank))
```

- Check Solution:

```
head(TopPopDesc)
## # A tibble: 6 x 5
##   State          Pop2020 NumChange2020 PercentChange2020 StateRank
##   <chr>            <dbl>         <dbl>             <dbl> <chr>
## 1 North Carolina 10439388        903905               9.5 9
## 2 Georgia        10711908       1024255              10.6 8
## 3 Ohio           11799448        262944               2.3 7
## 4 Illinois       12812508        -18124              -0.1 6
## 5 Pennsylvania   13002700        300321               2.4 5
## 6 New York       20201249        823147               4.2 4

head(LowPopDesc)
## # A tibble: 6 x 5
##   State                Pop2020 NumChange2020 PercentChange2020 StateRank
##   <chr>                  <dbl>         <dbl>             <dbl> <chr>
## 1 District of Columbia  689545         87822              14.6 X
## 2 Wyoming               576851         13225               2.3 50
## 3 Vermont               643077         17336               2.8 49
## 4 Alaska                733391         23160               3.3 48
## 5 North Dakota          779094        106503              15.8 47
## 6 South Dakota          886667         72487               8.9 46
```

bonus2: combine filter() and arrange()

- Challenge: In one command, from the df: Census2020Sub, Filter the states that are between the rank of 10 and 30 and Arrange by variable Pop2020

- Code:

```
Census2020Sub$StateRank <- as.numeric(Census2020Sub$StateRank, na.rm = TRUE)

Census2020Bonus2 <- Census2020Sub %>%
  filter(StateRank >= 10 & StateRank <= 30) %>%
  arrange(desc(Pop2020))
```

- Check Solution:

```
glimpse(Census2020Bonus2)
## Rows: 21
## Columns: 5
## $ State             <chr> "Michigan", "New Jersey", "Virginia", "Washington", ~
## $ Pop2020           <dbl> 10077331, 9288994, 8631393, 7705281, 7151502, 702991~
## $ NumChange2020     <dbl> 193691, 497100, 630369, 980741, 759485, 482288, 5647~
## $ PercentChange2020 <dbl> 2.0, 5.7, 7.9, 14.6, 11.9, 7.4, 8.9, 4.7, 7.0, 2.8, ~
## $ StateRank         <dbl> 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, ~
```

mutate(): adds new variables that are functions of existing variables

- Create, modify, and delete columns — mutate • dplyr (tidyverse.org)
- mutate adds new variables to your dataset and preserves existing ones
  - Note, transmute() does the same except drops existing variables, and will only keep new variables
  - Newdf <- df %>% mutate(new_variable = (existing variable2- existing variable1))
  - Newdf <- df %>% mutate(new_variable = (existing variable2 * 5))
- Exercise goal: Recreate the initial 2010 Population column by using Pop2020 and NumChange2020 from df: Census2020Sub
- See here for useful creation functions to assist with creating new variables: 5.5.1 Useful creation functions

United States® Census Bureau

R

mutate(): adds new variables that are functions of existing variables

- Code:

```
Census2020Mutate <- Census2020Sub %>%
  mutate(Pop2010 = Pop2020-NumChange2020)
```

- Check Solution:

```
head(Census2020Mutate)
## # A tibble: 6 x 6
##   State      Pop2020 NumChange2020 PercentChange2020 StateRank  Pop2010
##   <chr>        <dbl>         <dbl>             <dbl>     <dbl>    <dbl>
## 1 Alabama    5024279        244543               5.1        24  4779736
## 2 Alaska      733391         23160               3.3        48   710231
## 3 Arizona    7151502        759485              11.9        14  6392017
## 4 Arkansas   3011524         95606               3.3        33  2915918
## 5 California 39538223       2284267               6.1         1 37253956
## 6 Colorado   5773714        744518              14.8        21  5029196
```

United States® Census Bureau

R

summarize(): used for aggregation; reduces multiple values down to a single summary

- summarize each group to fewer rows — summarize • dplyr (tidyverse.org)
- summarize will create a new data frame based on the conditions specified.
  - Newdf <- df %>% summarize(new_variable = sum(existing_variable))
  - Newdf <- df %>% summarize(new_variable = mean(existing_variable))
- Exercise goal: Summarize the total population of the United States, both in 2020 and 2010, using Pop2020 and the newly created Pop2010. Use df: Census2020Mutate built from the last step
  - Bonus, rerun the above code with mean instead of sum

United States® Census Bureau

R

summarize(): used for aggregation; reduces multiple values down to a single summary

- Code:

```
Census2020PopSum <- Census2020Mutate %>%
  summarize(Total2020 = sum(Pop2020))

Census2010PopSum <- Census2020Mutate %>%
  summarize(Total2010 = sum(Pop2010))
```

- Check Solution:

```
Census2020PopSum
## # A tibble: 1 x 1
##   Total2020
##       <dbl>
## 1 331449281

Census2010PopSum
## # A tibble: 1 x 1
##   Total2010
##       <dbl>
## 1 308745538
```

bonus3: calculate the mean

- Replace sum with mean in the previous code and see what happens:

```
Census2020PopMean <- Census2020Mutate %>%
  summarize(Total2020 = mean(Pop2020))

Census2010PopMean <- Census2020Mutate %>%
  summarize(Total2010 = mean(Pop2010))
```

- Check Solution:

```
Census2020PopMean
## # A tibble: 1 x 1
##   Total2020
##       <dbl>
## 1  6499006.

Census2010PopMean
## # A tibble: 1 x 1
##   Total2010
##       <dbl>
## 1  6053834.
```

bonus4: calculate the difference of the sum and mean population between 2020 and 2010

```
Census2020PopSum - Census2010PopSum

##    Total2020
## 1   22703743

Census2020PopMean - Census2010PopMean

##    Total2020
## 1   445171.4
```

United States® Census Bureau

bonus5: calculate the sum of large states

```
PopAboveLimitSum <- PopAboveLimit %>%
  summarize(TotalLarge2020 = sum(Pop2020))

PopAboveLimitSum
```

```
## # A tibble: 1 x 1
##   TotalLarge2020
##            <dbl>
## 1      179266447
```

United States® Census Bureau

bonus6: calculate the sum of small states

```
PopBelowLimitSum <- PopBelowLimit %>%
  summarize(TotalSmall2020 = sum(Pop2020))

PopBelowLimitSum
```

```
## # A tibble: 1 x 1
##   TotalSmall2020
##            <dbl>
## 1      152182834
```

United States® Census Bureau

mutate(), group_by() and summarize(): aggregate by group

- Group by one or more variables — group_by • dplyr (tidyverse.org)
- Group_by is an important helper with summarize. Without using group_by, your command will summarize the entire set of data. When you add in group_by, you can summarize by smaller groups.
  - Newdf <- df %<% mutate(new_variable1 = case_when(X > y ~ “category1”,
    X < y ~ “category2”)) %>%
    group_by (new_variable1) %>%
    summarize(new_variable2 = sum(existing_variable))
- Exercise goal: Add a new variable to df: Census2020Mutate using mutate, that groups states by size. Use group_by with the new size variable, and summarize the sum of the large and small states
- Challenge: On the df: Census2020Mutate, use mutate, group_by, and summarize to classify states by big or small using Pop2020. Then use group_by to group by size, and summarize the total population in each group. Reminder, big is classified by > 9999999 and small is classified by <= 9999999.

United States® Census Bureau

R

mutate(), group_by() and summarize(): aggregate by group, example 1

- Code:

```
Census2020Size <- Census2020Mutate %>%
  mutate(size = case_when(Pop2020 > 9999999 ~ 'Big',
                          Pop2020 <= 9999999 ~ 'Small')) %>%
  group_by(size) %>%
  summarize(sizetot = sum(Pop2020))
```

- Check Solution:

```
Census2020Size
## # A tibble: 2 x 2
##   size  Total2020
##   <chr>     <dbl>
## 1 Big   179266447
## 2 Small 152182834
```

United States® Census Bureau

mutate(), group_by() and summarize(): aggregate by group, example 2

- Code:

```
Census2020Growth <- Census2020Mutate %>%
  mutate(growth = case_when(NumChange2020 > 0 ~ 'growth',
                            NumChange2020 <= 0 ~ 'decline')) %>%
  group_by(growth) %>%
  summarize(changetot = sum(Pop2020))
```

- Check Solution:

```
Census2020Growth
## # A tibble: 2 x 2
##   growth     change
##   <chr>       <dbl>
## 1 decline  17567503
## 2 growth  313881778
```

count(): count the number of observations in a group

- Count observations by group — count • dplyr (tidyverse.org)
- Count lets you quickly count the unique values of one or more variables. It can be used alone or paired with other commands
- Newdf <- df %>% count(variable)
- Exercise goal: Perform the same mutation from the bonus exercise, creating two size categories of Big and Small. Then, Count how many states were included in each of the two groups that separated state by size using df: Census2020Mutate

count(): count the number of observations in a group

- Code:

```
Census2020Mutate %>%
  mutate(size = case_when(Pop2020 > 9999999 ~ 'Big',
                          Pop2020 <= 9999999 ~ 'Small')) %>%
  group_by(size) %>%
  count()
```

- Check Solution:

```
## # A tibble: 2 x 2
## # Groups:   size [2]
##   size      n
##   <chr> <int>
## 1 Big      10
## 2 Small    41
```

count(): count the number of observations in a group

- Code, try it a different way:

```
Census2020Mutate %>%
  mutate(size = case_when(Pop2020 > 9999999 ~ 'Big',
                          Pop2020 <= 9999999 ~ 'Small')) %>%
  count(size)
```

- Check Solution:

```
## # A tibble: 2 x 2
##   size      n
##   <chr> <int>
## 1 Big      10
## 2 Small    41
```

Take a Break, 1 hour

join: join two tables together

- Mutating joins — mutate-joins • dplyr (tidyverse.org)
- 13 Relational data | R for Data Science (had.co.nz)
- There are four primary join options to add columns from y to x, matching rows based on a key variable

- `inner_join()` : includes all rows in `x` and `y` .
- `left_join()` : includes all rows in `x` .
- `right_join()` : includes all rows in `y` .
- `full_join()` : includes all rows in `x` or `y` .

  - Newdf <- df %>% type_join(df2)
  - Newdf <- df %>% type_join(df2, by = key)

- Exercise goal: Join the 2020 Census Population dataset (df: Census2020Sub) and the 2019 Census Population dataset (df: 2019Pop) by state. Then, join the 2019 ACS Poverty dataset (df: 2019Poverty) by state

- Extra Information:

Filtering joins match observations in the same way as mutating joins, but affect the observations, not the variables. There are two types:

- `semi_join(x, y)` **keeps** all observations in `x` that have a match in `y` .
- `anti_join(x, y)` **drops** all observations in `x` that have a match in `y` .

join: join separate tables together

- Code:
- step 1: Join 2020 with the first ACS dataset, 2019 Population, by state

```
CensusData1 <- left_join(Census2020Sub, Census2019, by = "State")

colnames(CensusData1)
## [1] "State"                "Pop2020"              "NumChange2020"
## [4] "PercentChange2020" "StateRank"            "Estimate"
```

- step 2: Join the new dataset with the second ACS dataset, 2019 Poverty, by state. Generic "estimate" name will be a problem, also need to rename this field to something specific

```
CensusData1 <- CensusData1 %>%
  rename(PopEstimate2019 = Estimate)

CensusData2 <- left_join(CensusData1, Poverty2019, by = "State")
```

- Check Solution:

```
colnames(CensusData2)
## [1] "State"                "Pop2020"              "NumChange2020"
## [4] "PercentChange2020" "StateRank"            "PopEstimate2019"
## [7] "PovertyStatus"        "BelowPoverty"         "AbovePoverty"
```

cbind(): bind columns together

- Code:

```
CensusData3 <- as.data.frame(Census2019) %>%
  filter(!State  %in% c('Puerto Rico')) %>%
  rename(StateDrop = State, PopEstimate2019 = Estimate)

CensusData3 <- cbind(Census2020Sub, CensusData3) %>%
  select(-c('StateDrop'))

Poverty2019ACS <- as.data.frame(Poverty2019) %>%
  filter(!State  %in% c('Puerto Rico')) %>%
  rename(StateDrop = State)

CensusData3 <- cbind(CensusData3, Poverty2019ACS) %>%
  select(-c('StateDrop'))
```

- Check Solution:

```
colnames(CensusData2)
## [1] "State"              "Pop2020"           "NumChange2020"
## [4] "PercentChange2020" "StateRank"         "PopEstimate2019"
## [7] "PovertyStatus"     "BelowPoverty"      "AbovePoverty"

colnames(CensusData3)
## [1] "State"              "Pop2020"           "NumChange2020"
## [4] "PercentChange2020" "StateRank"         "PopEstimate2019"
## [7] "PovertyStatus"     "BelowPoverty"      "AbovePoverty"
```

United States® Census Bureau

rbind(): bind rows together

- Code:

```
Census2020 <- CensusData1 %>%
  select(State, Pop2020) %>%
  rename(Pop = Pop2020) %>%
  mutate(year = "2020")

Census2019 <- CensusData1 %>%
  select(State, PopEstimate2019) %>%
  rename(Pop = PopEstimate2019) %>%
  mutate(year = "2019")

Census2year <- rbind(Census2020, Census2019)
```

- Check Solution:

```
colnames(Census2year)
## [1] "State" "Pop"   "year"

rownames(Census2year)
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"
##  [13] "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"
##  [25] "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36"
##  [37] "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"
##  [49] "49"  "50"  "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"
##  [61] "61"  "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"  "71"  "72"
##  [73] "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"  "81"  "82"  "83"  "84"
##  [85] "85"  "86"  "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"
##  [97] "97"  "98"  "99"  "100" "101" "102"
```

United States® Census Bureau

mutate(): adds new variables that are functions of existing variables

- Create, modify, and delete columns — mutate • dplyr (tidyverse.org)
- Mutate adds new variables to your dataset and preserves existing ones. Transmute() does the same except drops existing variables, and will only keep new variables
  - Newdf <- df %>% mutate(new_variable = dense_rank(existing variable1))
- Exercise goal: Create state ranking based on PovertyStatus variable in df: CensusData2, and filter for state with a poverty rank less than or equal to 10
- See here for useful creation functions to assist with creating new variables: 5.5.1 Useful creation functions

United States® Census Bureau

R

mutate(): adds new variables that are functions of existing variables

- Code:

```
CensusDataRanked <- CensusData2 %>%
  mutate(PovertyRank = dense_rank(desc(BelowPoverty))) %>%
  mutate(PovertyPercent = 100 * (BelowPoverty / PovertyStatus)) %>%
  filter(PovertyRank <= 10)
```

- Check Solution:

```
glimpse(CensusDataRanked)
## Rows: 10
## Columns: 11
## $ State             <chr> "California", "Florida", "Georgia", "Illinois", "Mic~
## $ Pop2020           <dbl> 39538223, 21538187, 10711908, 12812508, 10077331, 20~
## $ NumChange2020     <dbl> 2284267, 2736877, 1024255, -18124, 193691, 823147, 9~
## $ PercentChange2020 <dbl> 6.1, 14.6, 10.6, -0.1, 2.0, 4.2, 9.5, 2.3, 2.4, 15.9
## $ StateRank         <dbl> 1, 3, 8, 6, 10, 4, 9, 7, 5, 2
## $ PopEstimate2019   <dbl> 39283497, 20901636, 10403847, 12770631, 9965265, 195~
## $ PovertyStatus     <dbl> 38733295, 21048884, 10332523, 12373209, 9772151, 189~
## $ BelowPoverty      <dbl> 4552837, 2664772, 1373909, 1420542, 1269062, 2467006~
## $ AbovePoverty      <dbl> 34180458, 18384112, 8958614, 10952667, 8503089, 1646~
## $ PovertyRank       <int> 1, 3, 9, 7, 10, 4, 8, 6, 5, 2
## $ PovertyPercent    <dbl> 11.75432, 12.65992, 13.29694, 11.48079, 12.98652, 13~

CensusDataRanked$PovertyRank
##  [1]  1  3  9  7 10  4  8  6  5  2

CensusDataRanked$PovertyPercent
##  [1] 11.75432 12.65992 13.29694 11.48079 12.98652 13.03054 13.59045 13.06822
##  [9] 12.02318 13.62770

CensusDataRanked$State
##  [1] "California"     "Florida"        "Georgia"        "Illinois"
##  [5] "Michigan"       "New York"       "North Carolina" "Ohio"
##  [9] "Pennsylvania"   "Texas"
```

ggplot: steps to create a basic visualization

- Create Elegant Data Visualisations Using the Grammar of Graphics • ggplot2 (tidyverse.org)
- Use ggplot to create visualizations. The seven parameters in the below template compose the grammar of graphics, a formal system for building plots. The grammar of graphics is based on the insight that you can uniquely describe *any* plot as a combination of a dataset, a geom, a set of mappings, a stat, a position adjustment, a coordinate system, and a faceting scheme.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

DATA: Data set used for visualization

GEOM_FUNCTION: How should data be graphically displayed

Options: bar, point, line, boxplot, path, smooth, histogram

MAPPINGS: Mapping the data to geom properties (ie aesthetics)

STAT: The statistical transformation used on the data

POSITION: Position adjustment such as width and height

COORDINATE_FUNCTION: Desired coordinate system, creates different position scale for x and y variables

FACET_FUNCTION: Split graph into subplots

United States® Census Bureau

R

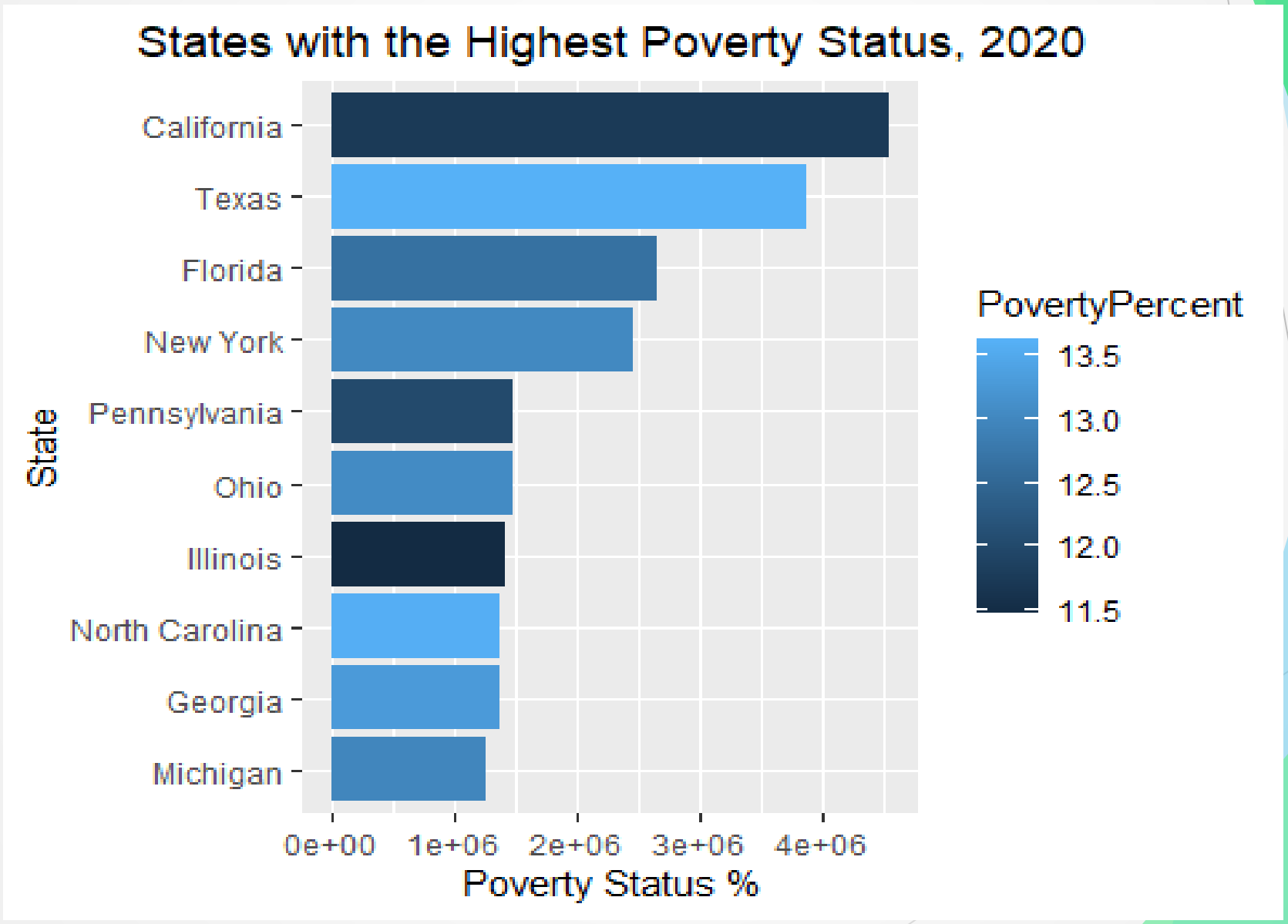
ggplot: create a basic bar chart

```
ggplot(CensusDataRanked) +
  geom_bar(mapping = aes(x = reorder(State,BelowPoverty),
                         y = BelowPoverty,
                         fill = PovertyPercent),
           stat = 'identity') +
  labs(title = "States with the Highest Poverty Status, 2020",
       x = "State",
       y = "Poverty Status %") +
  theme(plot.title = element_text (hjust = 0.5)) +
  coord_flip()
```

United States® Census Bureau

R

ggplot: create a basic bar chart



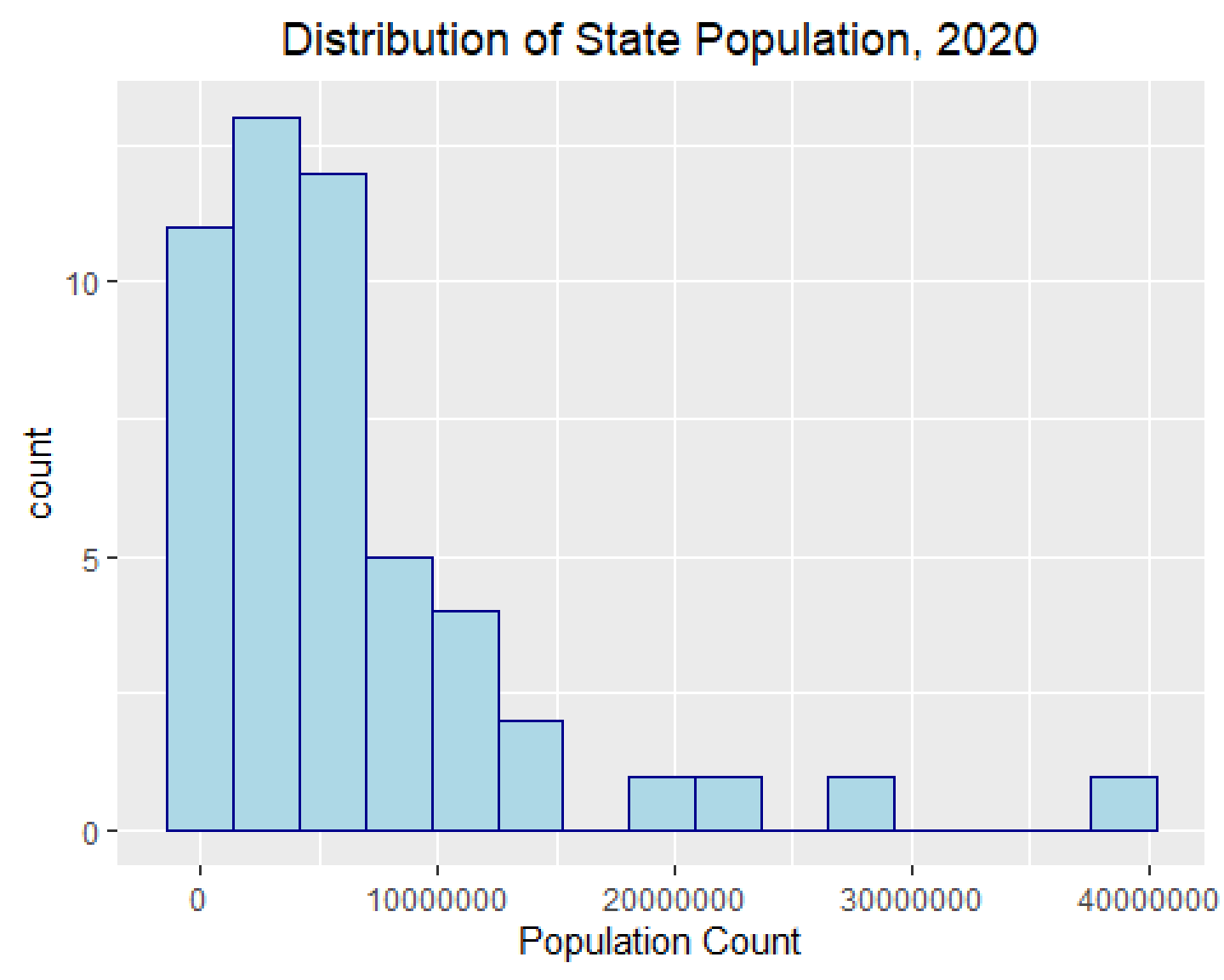
ggplot: create a histogram

```
options(scipen = 999) # Disable scientific notation

ggplot(CensusData2, aes(x = Pop2020)) +
  geom_histogram(fill ='light blue',
                 col = 'dark blue',
                 bins = 15) +
  labs(title = "Distribution of State Population, 2020",
     x = "Population Count") +
  theme(plot.title = element_text (hjust = 0.5))
```

United States® Census Bureau

ggplot: create a histogram



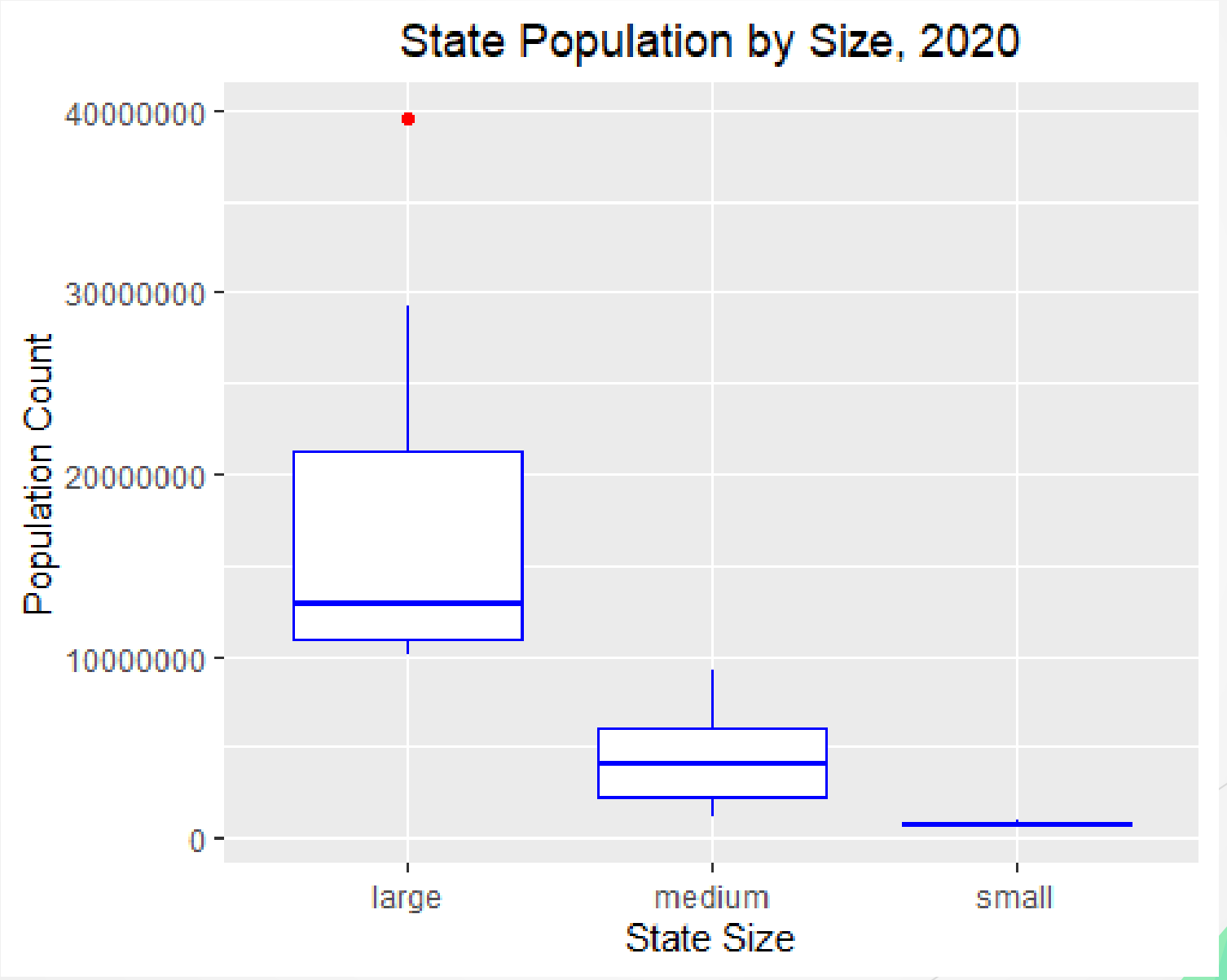
ggplot: create a boxplot

```
CensusData2Size <- CensusData2 %>%
  mutate(changedir = case_when(NumChange2020 > 0 ~ 'increase',
                               NumChange2020 < 0 ~ 'decrease')) %>%
  mutate(size_bin = case_when(Pop2020 < 1000000 ~ 'small',
                              Pop2020 >= 1000000 & Pop2020 <= 10000000 ~ 'medium',
                              Pop2020 > 10000000 ~ 'large')) %>%
  mutate(SizeRank = dense_rank(desc(Pop2020)))
```

```
ggplot(CensusData2Size, aes(x = size_bin, y = Pop2020)) +
  geom_boxplot(color = 'blue', outlier.color = 'red') +
  labs(title = "State Population by Size, 2020",
       x = "State Size",
       y = "Population Count") +
  theme(plot.title = element_text (hjust = 0.5))
```

United States® Census Bureau

ggplot: create a boxplot



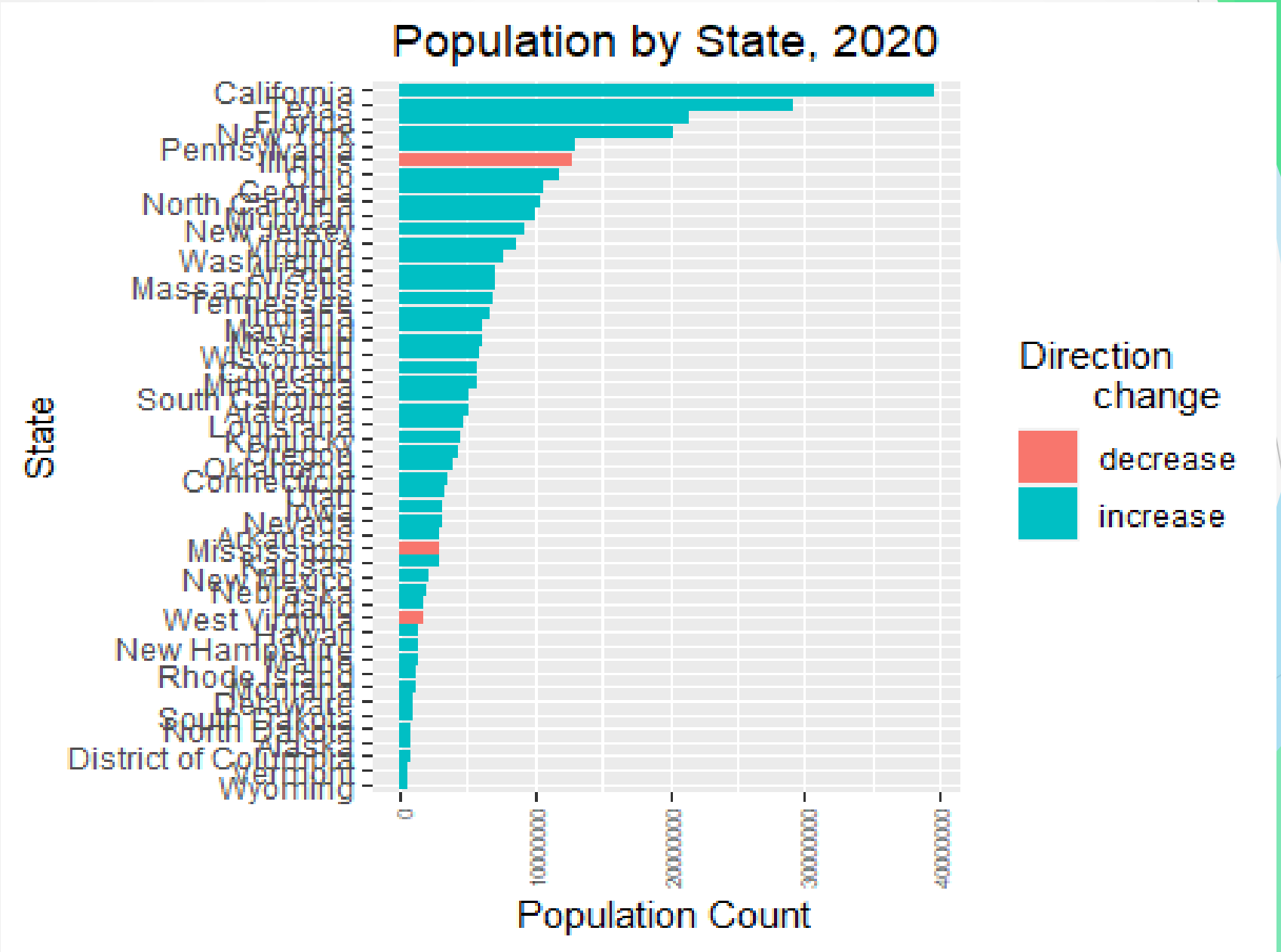
United States® Census Bureau

ggplot: create a bar chart

```
ggplot(CensusData2Size, aes(x = reorder(State, Pop2020),
                            y = Pop2020, fill = changedir)) +
  geom_col() +
  labs(title = "Population by State, 2020",
       x = "State",
       y = "Population Count",
       fill = "Direction
       change") +
  theme(axis.text.x = element_text(angle=90,hjust=1,vjust=0.5, size = 6))+
  theme(plot.title = element_text (hjust = 0.5)) +
  coord_flip()
```

United States® Census Bureau

ggplot: create a bar chart

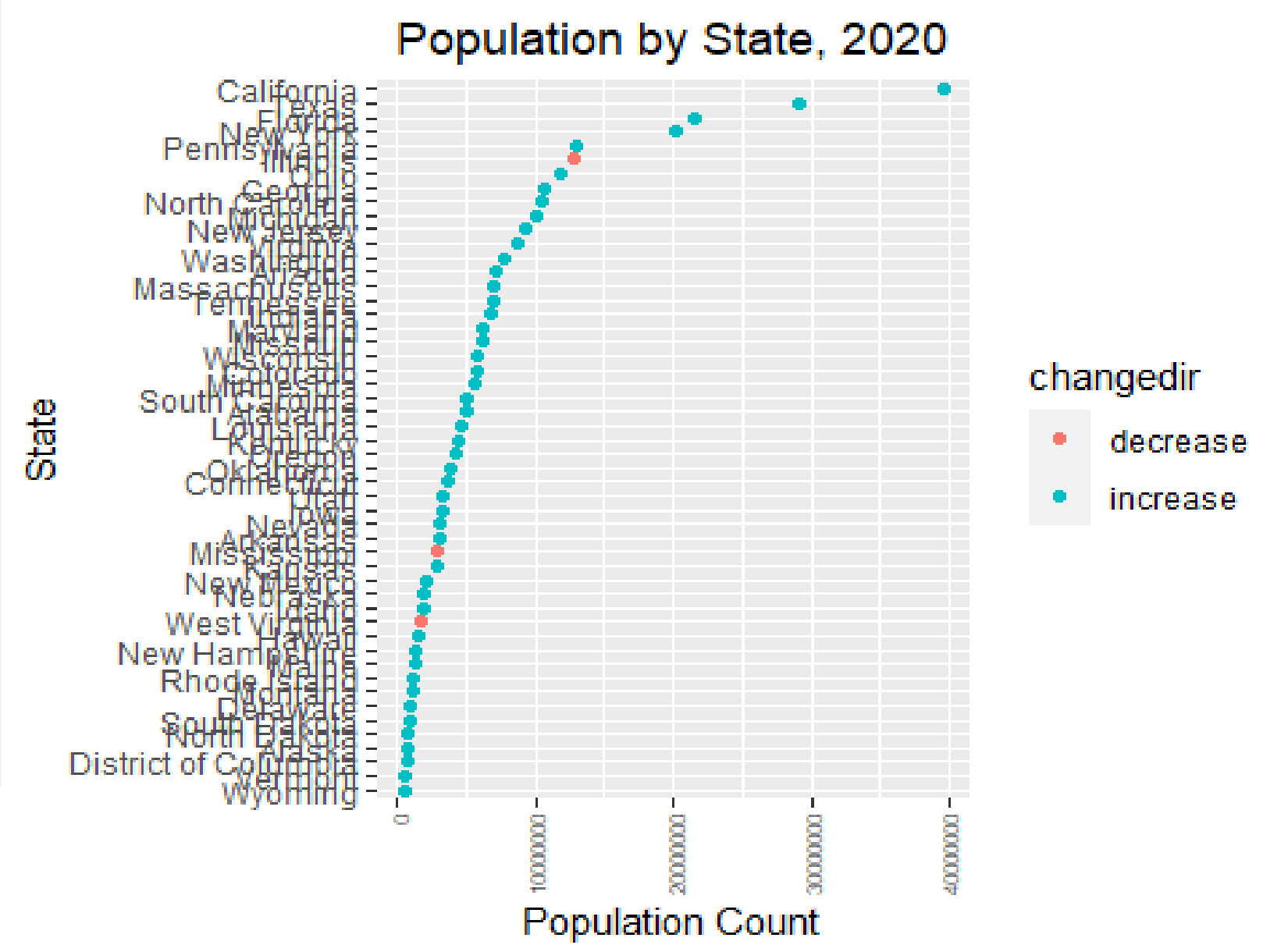


ggplot: create a scatterplot

```
ggplot(CensusData2Size, aes(x = reorder(State, Pop2020),
                            y = Pop2020, color = changedir)) +
  geom_point() +
  labs(title = "Population by State, 2020",
       x = "State",
       y = "Population Count") +
  theme(axis.text.x = element_text(angle=90,hjust=1,vjust=0.5, size = 6))+
  theme(plot.title = element_text (hjust = 0.5)) +
  coord_flip()
```

United States® Census Bureau

ggplot: create a scatterplot



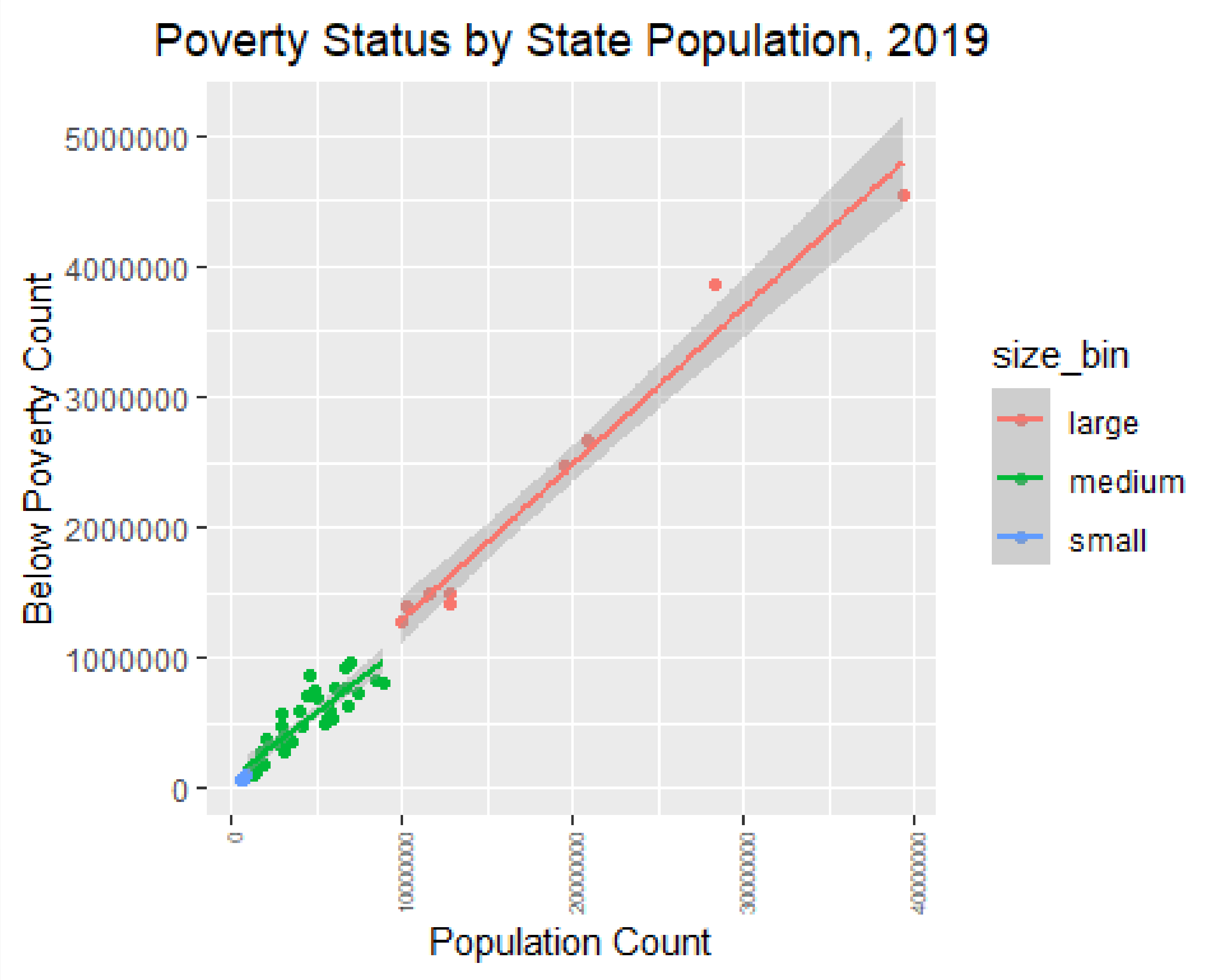
United States® Census Bureau

ggplot: create a scatterplot with regression line

```
ggplot(CensusData2Size, aes(x = PopEstimate2019,
                            y = BelowPoverty, color = size_bin)) +
  geom_point() +
  geom_smooth(method = 'lm') +
  labs(title = "Poverty Status by State Population, 2019",
       x = "Population Count",
       y = "Below Poverty Count") +
  theme(axis.text.x = element_text(angle=90,hjust=1,vjust=0.5, size = 6)) +
  theme(plot.title = element_text (hjust = 0.5))
```

United States® Census Bureau

ggplot: create a scatterplot with regression line

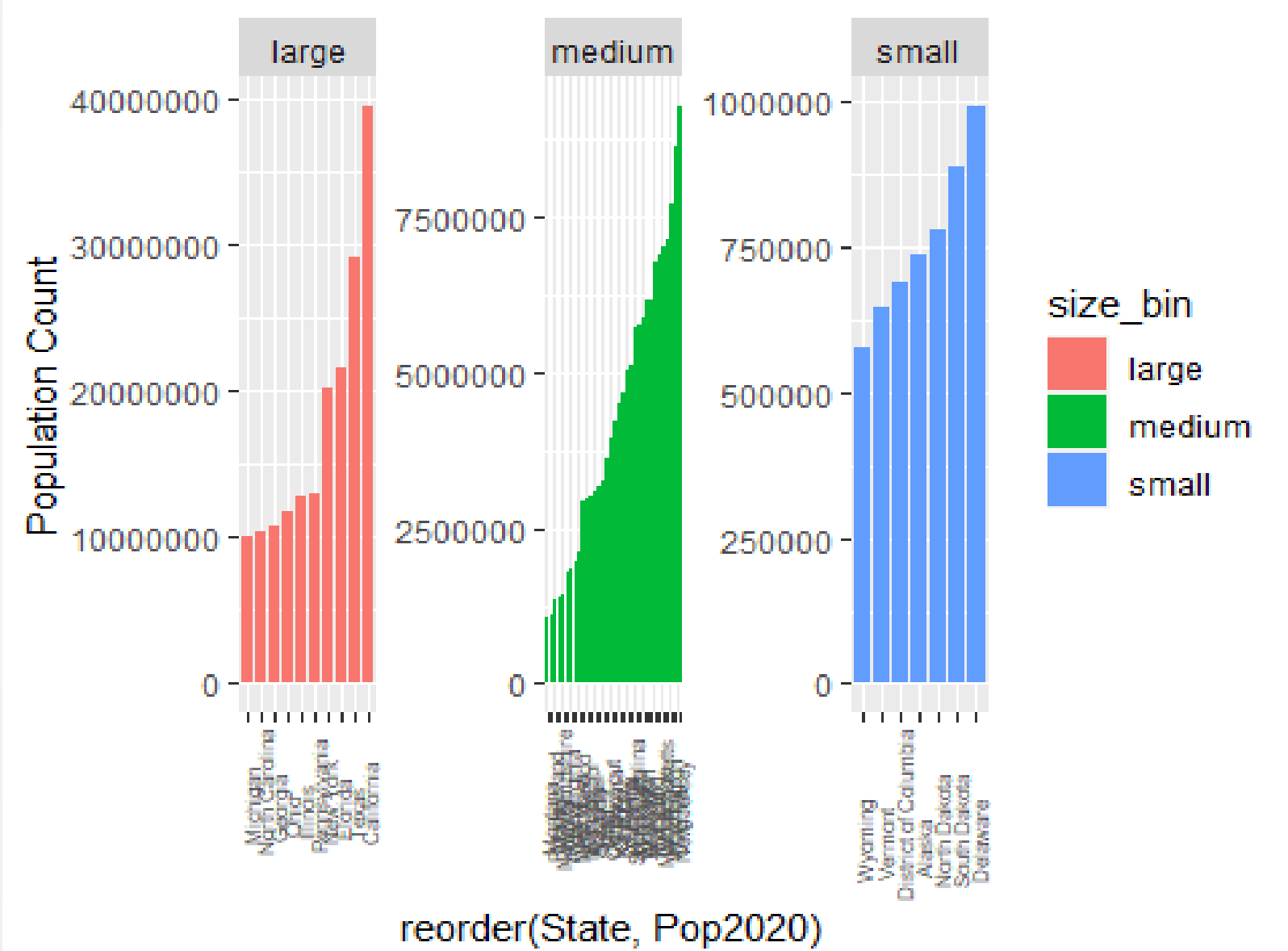


ggplot: create a bar chart with facets

```
ggplot(CensusData2Size, aes(x = reorder(State, Pop2020),
                            y=Pop2020, fill = size_bin))+
  geom_col()+
  ylab("Population Count")+
  facet_wrap(~size_bin, scale = "free") +
  theme(axis.text.x = element_text(angle=90,hjust=.2,vjust=0.5, size = 6)) +
  theme(plot.title = element_text (hjust = 0.5))
```

United States® Census Bureau

ggplot: create a bar chart with facets



United States® Census Bureau

Take a Break, 10 minutes

R Markdown

- R Markdown (rstudio.com)
- Utilized for sharing deliverables to stakeholders in a presentation style format
- See CBRUG recording on using R Markdown
- rmarkdown-cheatsheet (rstudio.com)

- Demonstration

United States® Census Bureau

GitHub

- GitHub
- Enterprise-wide GitLab coming soon, free for all Census employees
- Utilized to archive and share code with others
- See CBRUG recording on using Git
- Git cheatsheet - GitHub Docs

- Demonstration

United States® Census Bureau

Extra credit

- Complete the extra credit assignment and sent it to me by December 16th, and you will be sent a prize!
- You can complete the .R file, or get bonus points if you use .Rmd to submit the assignment
  - read in a variable from tidycensus or excel
  - Use three of the commands for EDA to inspect your data
  - select
  - rename
  - filter
  - arrange
  - mutate
  - summarize
  - mutate, group_by and summarize
  - create workflow of your choice using the above verbs
  - create a visualization, properly formatted with labels

United States® Census Bureau

Census Team:

- Jessica Klein, ERD
- Cecile Murray, ERD
- John Lombardi, EWD
- Keith Savage, EID

Center for Applied Technology Team:

- Christopher Jackson, CAT Program Manager
- Kevin Schweickhardt, Operations Manager
- Mohammed Chizari, Data Scientist
- Saleem Shaik, Data Scientist
- Jimmy Pazouki, Systems Administrator
- Llewellyn Forbes, Operations Manager

Department of Commerce
U.S. Census Bureau
Room 1J250
4600 Silver Hill Road
Suitland, MD 20746
(301) 763 - 4300
cat@census.gov

United States® Census Bureau